Institute for Mathematical
Studies in the Social Sciences
Stanford University
Stanford, California
August 1, 1969

Reference: NASA Research Grant NGR-05-020-244

Status Report (January 1, 1969 - June 30, 1969)

Since the first year of the AID project has now ended, this report includes a review of the goals of the project and a progress report of these goals. A more detailed report of the activities of the months April through June is also included.

I. Review of Goals

The major goal of the project is to develop a system for teaching computer programming (in particular, the programming language AID) by means of computer-assisted instruction. The course is to be completely self-contained and is directed at students of about junior college ability. Students communicate with the computer through standard teletypes, which are connected to the computer by telephone lines. The computer, in its role as teacher, presents instructions and problems to the student by typing on his teletype, the student responds by typing on the same teletype, and the response is relayed to the computer and analyzed. After analyzing the student's response, the computer must then reply, correcting the student if necessary and presenting him with additional instruction. At the same time the computer must handle simultaneously large numbers of other students, who may be taking the same course, or who may be enrolled in completely independent courses (computer-based Russian, for example). The computer must work in a time-sharing mode.

The computer-assisted course may, of course, use supplementary materials such as reference manuals for the programming language being taught. Nevertheless,

the course, including the supplementary materials, should be self-contained in the sense that it is independent of lectures or other instruction and does not require the presence of an experienced teacher of programming.

II. Summary of the Year's Activities

In planning the system for teaching programming, we felt that the relatively mature students who would be enrolled in the course would be capable of making decisions about the course of study and also would be motivated by being permitted to make such decisions. Thus, no arbitrary remedial branching structure was incorporated in the system; in its place, the curriculum will advise students to review previous lessons, to do certain sets of practice or remedial exercises, or to skip lessons which may be either too easy or too difficult. The student may choose to follow the advice given in the lessons, or to ignore it, since he can control the sequence of problems by the use of a few simple "control commands."

Although no major branching structure is imposed on the student, there is a rather complex system of "micro" branching within each problem. After a problem statement is presented by the computer, the student may make a number of different kinds of responses; for example, he may request additional instruction; he may demand the correct answer; he may skip the problem and go on to the next; he may request any other problem in the course; he may, of course, type the correct answer; or he may type an incorrect response. For each case the computer provides a different response. Since the system can make fine discriminations between student responses, each problem takes on the aspect of a little "conversation." This conversation may develop into a dialogue of some length, depending upon the number of messages supplied by the writer.

As plans for the system developed, it became clear that the major goal implied two subsidiary goals: the development of the curriculum and the development of a set of programs which would interpret the lessons and interact with the students.

A. The curriculum. The course in AID programming was planned as a series of lessons presenting the fundamentals of computer programming in the context of an introduction to the programming language AID. AID[1] (Algebraic Interpretive Dialogue) is a high-level algebraic programming language with extensive interactive (or "conversational") abilities. This language is an adaptation for the PDP-10 computer of JOSS,[2] a language which was developed by RAND Corporation for use by scientists, engineers, etc., who needed a powerful, easy-to-learn tool capable of performing complex algebraic tasks.

It was assumed that students would have no previous experience with programming or computers, although a certain facility with algebra (as supplied by a good course in high school algebra) was necessary. The more esoteric algebraic concepts, such as hierarchy of operations, zero exponents, and conditional definition of functions, are reviewed in the course. Necessary concepts from logic are also taught in the course. Certain features of AID,

---

[1] See PDP-10 AID Programmer's Reference Manual, Digital Equipment Corporation, Maynard, Massachusetts, 1968.

[2] See Mark, S. L. and Armerding, G. W., The JOSS Primer. The RAND Corporation, Santa Monica, California, August 1967; Shaw, J. C., JOSS: Experience with an Experimental Computing Service for Users at Remote Typewriter Consoles. The RAND Corporation, Santa Monica, California, May, 1965.

such as transcendental functions and recursive definition of functions, are of little use to students who have not had an introduction to calculus or modern algebra; such lessons are optional and should be bypassed by most students. An outline of the AID course is given in Table 1.

In addition to the programmed lessons, a reference manual is available for student use, containing an outline and brief description of the lessons, additional programming problems to be done by the student, etc.

Of the 50 lessons which constitute the introductory course, 33 have been written and 23 coded; the first two lessons have been revised, and plans have been made to revise Lessons 3 to 20. This revision is based on a small-scale pilot study conducted during the spring of 1969. The preliminary version of the student manual has been written and was used by students; plans for a revision of the manual have been made.

B. The instructional system. In order to implement the planned curriculum, it was necessary to develop a lesson coding language, an interpreter program, and a supporting preprocessor system. The major activity of the year was the design of this set of programs and the production of a preliminary version of the system, referred to as the "interim system" in previous reports.

The interpreter ("teaching program") interacts with students at the time they are working on a lesson. This interpreter must be able to interpret the instructions given it by the curriculum writer and to use that information to determine which problem to give the student, how to analyze his response, and how to react properly to that response.

The instructions given to the interpreter by the writer must, of course, be in a simplified "code" which can be understood by the interpreter, hence

a necessary part of the instructional system is a rigorously defined, unambiguous, coding language. The coding language, which may be used by relatively inexperienced personnel, had to be easy to learn and easy to use and preferably related closely to the English language. In other words, the coding language had to be a genuine higher-level programming language.

The interpreter had to be written to interpret this coding language directly. However, since response time is such a critical concern in the design of any interactive system, it was desirable to have the coded lessons put through at least one intermediate translation to transform the code into a language more readily understood (or at least, more rapidly understood) by the interpreter. Thus, a system of intermediate translation, or "lesson preprocessing," was desirable. The lesson preprocessing may then be done before a student commences a lesson, perhaps the evening before, or perhaps months before, with a significant beneficial effect on the response time of the interpreter.

During the year the specifications for the instructional system were written, a preliminary version of the coding language was developed, and the programs for lesson preprocessing and for interpreting were written, debugged and used by students in the pilot study.

Based on experiences of coders and students, plans were made for a revision and extension of all components of the instructional system. The new instructional system is now being prepared; detailed description is given in Section IV of this report.

As the instructional system developed it became evident to all concerned that, although the first application of the system was to be the course in

AID programming, the system was equally well suited to teach any programming language, or indeed to teach almost any other subject amenable to computer-assisted instruction. The system is not well suited for drill-and-practice or for experimental use where a more rigid branching structure is desired. Nor is it equipped with the peripheral apparatus (audio, visual displays) necessary for subjects such as foreign languages or art. However, any subject which is primarily conceptual and which may be taught verbally, is an ideal candidate for computer-assisted instruction using this teaching system. An effort has been made throughout the design and implementation of the instructional system to maintain this generality.

III. The Pilot Study

During the Spring Quarter, 1969, there was a very small pilot study whose purpose was to supply information for meaningful revisions of the curriculum and the instructional system. Since this was the first trial of the system, the most useful information would be derived from students' reactions to the program. There was no plan to collect detailed data or to do any kind of statistical analysis of data. Ten students were enrolled in the course on a flexible time scheduling basis; some students were scheduled three sessions a week, others two, and others came only once a week, depending upon the wishes of the individual students. The students were allowed to use the course in whatever way they felt best; but they were restricted to taking not more than two lessons per session. Also, immediately after each session, they were to be interviewed for about 5 to 10 minutes.

The students completed anywhere from three to twenty lessons each, with about half of them getting as far as Lesson 20. In general, the students

who did fewer lessons did so because they spent less time on the lessons rather than because of any great difficulties with the material. In fact, the student who had the most difficulty with the course, and made the slowest progress in relation to the time spent, finished Lesson 13 by the end of the quarter and expressed regret that he hadn't been able to spend enough time to have completed the 20 available lessons.

Students were timed on several lessons in order to get a rough idea of the time which would be necessary for future students to complete the course. The average time per problem for different students ranged from about one minute per problem to three minutes per problem; the assignments for each lesson required about as much time as the lesson itself.

Extensive notes were taken during interviews with the students and were summarized in an anecdotal weekly report. Also, the responses to individual problems were tabulated and the percentages of correct and incorrect responses were calculated. The most frequent incorrect response to each problem was also tabulated.

The students were quite enthusiastic about the course and would have worked for several hours at a time had they not been restricted to taking no more than two lessons per session. Since most of the students' comments were about specific problems, there was no indication that a major revision of the curriculum is needed. The following are a few general observations based on students' comments and behavior.

Use of student controls. The student control commands, which were explained in detail in Lesson 1, were received with enthusiasm. (A control command is given by holding down the "CTRL" key while striking a letter key.)

The commands used were

Ctrl-H (used to request a hint)

Ctrl-T (used to request the answer)

Ctrl-S (skip to next problem)

Ctrl-G (used to get another problem or lesson. After the student types Ctrl-G he is asked to specify the lesson and problem he desires.)

Both Ctrl-H and Ctrl-T were used frequently, although there was noticeable tendency for students to use one or the other but not both. Ctrl-S was rarely used; in fact, several students were asked, at the end of lesson 3, what control commands were available and were not able to recall Ctrl-S.

Ctrl-G was used much less than anticipated. At the end of the pilot study, the students were queried about this; several students replied that they thought they would not be contributing fully to the experiment (the pilot study) if they skipped any of the lessons; a few students felt that they would not know what they had skipped and that it might be important to them in later lessons (this comment was made even in reference to reviews and self-tests in which there was an explicit statement that no new material would be presented and that it was perfectly acceptable to skip the entire lesson); only one student consistently chose to review previous lessons and he commented that he felt he simply repeated the same mistakes without achieving any noticeable gain in understanding.

Language confusion. Almost all students evidenced some confusion between the language they were learning (the AID programming language) and the language (English) used in the exposition. Part of this confusion undoubtedly arose because the AID language is a subset of English (AID commands are syntactically

correct English sentences that contain a verb, end with a period, etc.); although this is certainly not a complete explanation and it is obvious that the advantages of teaching an English-based programming language far outweigh the disadvantages even if it could be shown to be a significant factor in the language confusion.

Furthermore, a few students were also puzzled about which program they were using--the teaching program or the AID interpreter (which they used for doing assignments); one student tried to ask the AID interpreter for hints about an assigned programming problem. It is felt that some confusion between languages and between programs is almost inherent in the situation and no satisfactory way of dispelling the confusion has been found.

Constructed Responses to Multiple-Choice Problems

The multiple-choice problems used in the course consist of a problem statement or question and a list of possible answers, each of which is labeled with a letter. For example,

WHICH OF THESE ARE CORRECT AID COMMANDS?

A. TYPE 2 × 3.

B. PRINT 2 * 3.

C. TYPE 2 ** 3.

N. NONE OF THE ABOVE.

Students are expected to respond by typing a letter (or list of letters) corresponding to the correct answer (or answers).

There is a noticeable tendency for students to respond to certain multiple-choice problems by typing the answer itself rather than typing the corresponding letter. In the AID course, a response other than a single letter (or list of

letters) is treated as an error, and the message

PLEASE TYPE LETTERS ONLY

is given. This error message has been found to be remarkably ineffective; the probability that a student will repeat the same kind of error after receiving the above error message seems to be greater than one half, possible as much as three quarters.

The tendency to make the kind of error described above seems to be influenced by the following factors:

1. Answer length. If the number of characters in the answer choice is small (say, two to six characters), there is a strong tendency to type the answer itself.

2. Context. If the problem is preceded by several problems requiring constructed responses, the tendency to construct a response is somewhat increased. If the preceding constructed responses are closely related to the choices in the multiple-choice problem, there is an even stronger tendency to construct a response; for example, if the six preceding problems require 3-digit numbers as a response, and the choices in the multiple-choice problem are 3-digit numbers, there is a high probability of making an error.

3. Problem-solving strategy required. There seem to be two distinct kinds of problem-solving strategies used in producing the answer to a multiple-choice problem. One is a "mental construction" of the correct answer, followed by a search of the choices for that answer, and the other kind is a "feasibility-elimination" approach in which the student inspects the list of possible answers and chooses that which is most feasible, or eliminates those choices which are least feasible. (Generally, students working on a specific problem

will not switch from one strategy to another unless there is a compelling reason; for instance, a student will abandon a "feasibility-elimination" approach if several choices are equally feasible.) The strategy a student uses is influenced by the problem statement although there is some tendency for individual students to prefer one strategy over another. If the "mental construction" strategy is used, the student is more likely to produce an overt construction of the answer, thereby producing an "error."

4. Wording used in problem statement. The wording used in instructions to the student seems to have some effect on the tendency to give a constructed answer to a multiple-choice problem. In particular, use of the word "what" in the problem statement produces more errors than the word "which." For example, compare "What command causes AID to give N a value of 12?" with "Which command causes AID to give N a value of 12?"

One additional comment: Although the above remarks may imply that the error of constructing a response in answer to a multiple-choice question is a use-mention error, this may not be the case. There are a number of problems in the course which require a "partial construction" and there is an observable tendency in students to give a more complete answer than is required; for example, students tend to answer "Do Part 12" rather than "Do" in response to this problem:

COMPLETE THIS COMMAND TO

EXECUTE PROGRAM 12.

..... PART 12.

The error of constructing a more complete response than required is clearly not a use-mention error, and it seems to be closely related to the error of constructing a response to a multiple-choice problem.

Answer length, context, required strategy, and wording used in problem statement are not the only factors which contribute to the kind of use-mention error under consideration here; there are also individual factors, such as age and previous experience. However, the above four factors are the only curriculum-oriented factors which seem to have an effect.

IV. Other Activities of the Past Quarter

In addition to conducting a pilot study, the staff continued writing lessons, coding and debugging.

A major revision of the coding language was made and work on the time-shared version of the instructional system was started. The revised instructional system will be discussed in detail in the next progress report.

V. Activities Planned for the Next Reporting Period

During the next three months detailed plans for a revision of the curriculum will be made, some of the lessons will be rewritten and lesson coding (using the revised coding language) will commence.

A coders' manual for the new coding language will be written.

The major effort in the next months will be the preparation of the programs for the revised instructional system. The two major programs, the lesson driver and the lesson preprocessor, will be written first; supporting programs, such as the student enrollment program, the data collector, etc., will not be written until the major programs are completed.

Report submitted by

Richard C. Atkinson
Professor of Psychology

TABLE 1

OUTLINE

Computer-Assisted Instruction in Programming: AID

Lesson 1.	How to answer. How to erase. Control commands.
Lesson 2.	Signing on and off AID. The TYPE command. Arithmetic operators: + - * / . Decimal numbers.
Lesson 3.	Using AID for arithmetic. Use of parentheses. Order of arithmetic operations.
Lesson 4.	The operator ↑ for exponentiation. Order of operations. Scientific notation.
Lesson 5.	Variables. The SET command. Re-defining variables. The DELETE command used to delete variables.
Lesson 6.	Self-test.
Lesson 7.	Review.
Lesson 8.	The LET command (using function notation). Distinction between LET and SET. Distinction between use of a defined function and display of the formula for a function. Re-defining and deleting functions.
Lesson 9.	Some standard AID functions: IP(x), FP(x), SGN(x), SQRT(x).
Lesson 10.	Indirect steps. DO STEP .... DO STEP ... FOR .... Re-defining steps and deleting steps. TYPE STEP ....
Lesson 11.	Parts. DO PART .... DO PART .... FOR .... Deleting parts. TYPE PART.
Lesson 12.	The DEMAND command. DO PART ..., ... TIMES. Termination by refusal to answer a DEMAND command.

Lesson 13. Self-test.

Lesson 14. Review.

Lesson 15. Relations between numbers.
Relational symbols: < > <= >= = #
Number line.
The IF clause.

Lesson 16. Branching. The TO command.
TO STEP ...
TO PART ...

Lesson 17. Traces.

Lesson 18. The indirect use of DO.

Lesson 19. How to write and debug a program.

Lesson 20. Self-test.

Lesson 21. Review.

Lesson 22. The FORM statement.

Lesson 23. Loops.

Lesson 24. Loops with variable bounds.

Lesson 25. Loops compared with FOR clauses.

Lesson 26. Loops with a DEMAND command.

Lesson 27. Self-test.

Lesson 28. Review.

Lesson 29. Absolute value.

Lesson 30. Trigonometric functions: SIN(x), COS(x).

Lesson 31. EXP(x), LOG(x).

Lesson 32. Lists.

Lesson 33. Using loops with lists of numbers.

Lesson 34. Self-test.

Lesson 35. Review.

Lesson 36. More on loops. Decrementing counters. Formulas for exit condition.

Lesson 37. Iterative functions: SUM, PROD, MAX, MIN.

Lesson 38. Arrays.
LET S BE SPARSE.

Lesson 39. Conditional definition of functions.

Lesson 40. Recursion.

Lesson 41. Self-test.

Lesson 42. Review.

Lesson 43. AND, OR and NOT.
Truth tables.

Lesson 44. TV(x). The function FIRST.

Lesson 45. LET used to define propositions.

Lesson 46. More standard AID functions.

Lesson 47. More about lists and arrays.

Lesson 48. Self-test.

Lesson 49. Review.